河南大學
Henan University

I'm dying to play GTA 6

Contestant

范恒嘉	张帆	宋明贤
Hengjia Fan	Fan Zhang	Mingxian Song

目录

1 数据结构

1.1 DSU

```cpp
struct DSU
{
    vector<int> f, siz;

    DSU(int n) : f(n + 1), siz(n + 1, 1)
    {
        iota(f.begin(), f.end(), 0);
    }

    int find(int x)
    {
        if(x != f[x]) f[x] = find(f[x]);
        return f[x];
    }

    bool merge(int x, int y)
    {
        x = find(x);
        y = find(y);

        if (x == y)
            return false;

        if (siz[x] < siz[y])
            swap(x, y);

        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x)
    {
        return siz[find(x)];
    }

    bool connected(int x, int y)
    {
        return find(x) == find(y);
    }
};
```

1.2 Fenwick

```cpp
template <typename T>
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n) : n(n), a(n + 1) {}


    void add(int x, const T &v) {
        for (int i = x; i <= n; i += i & -i) {
            a[i] = a[i] + v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans = ans + a[i];
        }
        return ans;
    }

    T range(int l, int r) {
        return sum(r) - sum(l - 1);
    }

    T kth(T k) {
        int pos = 0;
        int logn = std::bit_width(a.size() - 1);
        for(int i = 1 << (logn - 1); i > 0; i >>= 1) {
            if(pos + i < a.size() && a[pos + i] < k) {
                k -= a[pos + i];
                pos += i;
            }
        }
        return pos + 1;
    }

};
```

1.3 ST 表

```cpp
template <typename T>
class SparseTable {

    // using func_type = function<T(const T &, const T &)>;

    vector<vector<T>> ST;
    // 更改操作，修改这一条语句
    static T op(const T &t1, const T &t2) { return max(t1, t2); }

    // func_type op;

    public:
    SparseTable(const vector<T> &v) {

        int n = v.size() - 1;

        int len = __lg(n);
        ST.assign(n + 1, vector<T>(len + 1, 0));

        for (int i = 0; i <= n; ++i) {
            ST[i][0] = v[i];
        }
        for (int j = 1; j <= len; ++j) {
            int pj = (1 << (j - 1));
            for (int i = 0; i + pj <= n; ++i) {
                ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
            }
        }
    }

    T query(int l, int r) {
        int q = __lg(r - l + 1);
        return op(ST[l][q], ST[r - (1 << q) + 1][q]);
    }
};
```

1.4 trie

```cpp

struct Trie {

    int tot;
    vector<vector<int>> nex;
    vector<int> cnt; // 以这个节点结尾的字符串的个数
    Trie() : nex(100001, vector<int>(26)), cnt(100001) {}
    Trie(int n, int m) : nex(n + 1, vector<int>(m)), cnt(n + 1) {}
    // n为树中最多会有多少个节点，m表示有多少种字符
    void insert(string s, int t = 1) {    // 插入字符串
        int p = 0;
        for (int i = 0; i < s.length(); i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) nex[p][c] = ++tot;    // 如果没有，就添加结点
            p = nex[p][c];
        }
        cnt[p] += t;
    }

    bool find(string s) {    // 查找字符串
        int p = 0;
        for (int i = 0; i < s.length(); i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) return 0;
            p = nex[p][c];
        }
        return cnt[p];
    }
};
```

1.5 动态开点线段树

```cpp
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
#define ls (id->lchild)
#define rs (id->rchild)
template<class Info, class Tag, class T = i64>
struct SegmentTree{
    struct Node {
        Node* lchild = nullptr;
        Node* rchild = nullptr;
        Info info;
        Tag tag;
    };
    T L, R;
    Node* root = nullptr;

    SegmentTree(T l, T r) : L(l), R(r) {
        root = new Node();
        (root->info).len = r - l + 1;
    }
    SegmentTree(T n) : SegmentTree(0, n) {}

    void apply(Node* id,const Tag &dx) {
        id->info.apply(dx);
        id->tag.apply(dx);
    }

    void pushdown(Node* id, T l, T r) {
        T mid = l + r >> 1;
        if (ls == nullptr) {
            ls = new Node();
            ls->info.len = mid - l + 1;
        }
        if (rs == nullptr) {
            rs = new Node();
            rs->info.len = r - mid;
        }
        apply(ls, id->tag);
        apply(rs, id->tag);
        id->tag = Tag();
    }

    void pushup(Node* id) {
        id->info = ls->info + rs->info;
    }

    void rangeUpdate(Node* id, T l, T r, T x, T y,const Tag &dx) {

        if (x <= l && y >= r) {
            apply(id, dx);
            return;
        }

        T mid = l + r >> 1;

        pushdown(id, l, r);

        if (x <= mid) {
            rangeUpdate(ls, l, mid, x, y, dx);
        }
        if (y > mid) {
            rangeUpdate(rs, mid + 1, r, x, y, dx);
        }
        pushup(id);
    }
    void modify(Node* id, T l, T r, T pos, const Info &val) {
        if (l == r) {
             id->info = val;
             return;
        }
        pushdown(id, l, r);
        T mid = l + r >> 1;
        if (pos <= mid) {
            modify(ls, l, mid, pos, val);
        } else {
            modify(rs, mid + 1, r, pos, val);
        }
        pushup(id);
    }


    Info rangeQuery(Node* id, T l, T r, T x, T y) {
        if (l >= x && r <= y) {
            return id->info;
        }
        Info res;
        T mid = l + r >> 1;

        pushdown(id, l, r);

        if (x <= mid) {
            res = res + rangeQuery(ls, l, mid, x, y);
        }
        if (y > mid) {
            res = res + rangeQuery(rs, mid + 1, r, x, y);
        }
        return res;
    }



    void rangeUpdate(T x, T y, const Tag &dx) {
        rangeUpdate(root, L, R, x, y, dx);
    }
    void modify(T pos, const Info &val) {
        modify(root, L, R, pos, val);
    }
    Info rangeQuery(T x, T y) {
        return rangeQuery(root, L, R, x, y);
    }
};
#undef ls
#undef rs
struct Tag {
    i64 add = 0;
    void apply(const Tag &dx) {
        add += dx.add;
    }
};

struct Info{
    i64 sum = 0;
    i64 len = 0;

    void apply(const Tag &dx) {
        sum += (dx.add * len);
    }
};
Info operator+(const Info a, const Info b) {
    Info res;
    res.sum = a.sum + b.sum;
    res.len = a.len + b.len;
    return res;
}



void solve() {
    int n, m;
    cin >> n >> m;
    SegmentTree<Info, Tag> seg(1, n);
    while(m--) {
        int op;
        cin >> op;
        if (op == 1) {
            int l, r, k;
            cin >> l >> r >> k;
            Tag tag;
            tag.add = k;
            seg.rangeUpdate(l, r, tag);
            // cout << 1 << endl;
        } else {
            int l, r;
            cin >> l >> r;
            cout << seg.rangeQuery(l, r).sum << '\n';
            // cout << 2 << endl;
        }
    }





}


int main() {
    std::ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

    int T = 1;
    //cin >> T;
    while (T--) solve();

    return 0;
}
```

1.6 区间修改线段树

```cpp
//线段树，区间修改，区间查询
//https://www.luogu.com.cn/problem/P3372
template<typename Info, typename Tag>
struct SegmentTree {
#define ls (id<<1)
#define rs (id<<1|1)
    SegmentTree() = default;
    SegmentTree(int n) : n(n), info(n << 2), tag(n << 2), len(n << 2) {}
    SegmentTree(const SegmentTree<Info, Tag> &o) : n(o.n), info(o.info), tag(o.tag)
    {}
    template<typename T>
    SegmentTree(const std::vector<T> &init) : SegmentTree((int)init.size()) {
        auto build = [&](auto self, int id, int l, int r) ->void {
            len[id] = r - l + 1;
            if(l == r) {
                info[id] = init[l];
                return;
            }
            int mid = (l + r) / 2;
            self(self, ls, l, mid);
            self(self, rs, mid + 1, r);
            pushup(id);
        };
        build(build, 1, 0, n - 1);
    }
    void apply(int id, const Tag &dx) {
        info[id].apply(dx, len[id]);
        tag[id].apply(dx);
    }
    void pushup(int id) {
        info[id] = info[ls] + info[rs];
    }
    void pushdown(int id) {
        apply(ls, tag[id]);
        apply(rs, tag[id]);
        tag[id] = Tag();
    }

    void modify(int id, int l, int r, int pos, const Info &val) {
        if(l == r) {
            info[id] = val;
            return;
        }
        pushdown(id);
        int mid = (l + r) / 2;
        if(pos <= mid) {
            modify(ls, l, mid, pos, val);
        } else {
            modify(rs, mid + 1, r, pos, val);
        }
        pushup(id);
    }

    void rangeUpdate(int id, int l, int r, int x, int y, const Tag &dx) {
        if(x <= l && r <= y) {
            apply(id, dx);
            return;
        }
        int mid = (l + r) / 2;
        pushdown(id);
        if(x <= mid) {
            rangeUpdate(ls, l, mid, x, y, dx);
        }
        if(y > mid) {
            rangeUpdate(rs, mid + 1, r, x, y, dx);
        }
        pushup(id);
    }
    Info rangeQuery(int id, int l, int r, int x, int y) {
        if(x <= l && r <= y) {
            return info[id];
        }
        int mid = (l + r) / 2;
        pushdown(id);
        Info res;
        if(x <= mid) {
            res = res + rangeQuery(ls, l, mid, x, y);
        }
        if(y > mid) {
            res = res + rangeQuery(rs, mid + 1, r, x, y);
        }
        return res;
    }


    void rangeUpdate(int l, int r, const Tag &dx) {
        rangeUpdate(1, 0, n - 1, l, r, dx);
    }
    void update(int pos, const Tag &dx) {
        rangeUpdate(pos, pos, dx);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n - 1, l, r);
    }
    Info query(int pos) {
        return rangeQuery(pos, pos);
    }

    void modify(int pos, const Info &val) {
        return modify(1, 0, n - 1, pos, val);
    }
#undef ls
#undef rs
    int n;
    std::vector<Info> info;
    std::vector<Tag> tag;
    std::vector<int> len;
};

constexpr i64 INF = 4E18;
i64 mod = 1e9 + 7;
struct Tag {
    i64 add = 0;
    i64 mul = 1;
    void apply(const Tag &dx) {
        mul = (mul * dx.mul) % mod;
        add = (add * dx.mul + dx.add) % mod;
    }
};

struct Info {
    i64 sum = 0;
    Info() {}
    Info(i64 x) : sum(x) {};
    void apply(const Tag &dx, const int &len) {
        sum = (sum * dx.mul + dx.add * len) % mod;
    }
};

Info operator+(const Info &x, const Info &y) {
    Info res;
    res.sum = (x.sum + y.sum) % mod;
    return res;
}
```

1.7 波纹疾走树

```
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
```

```cpp
struct BitRank {
    // block 管理一行一行的bit
    std::vector<unsigned long long> block;
    std::vector<unsigned int> count;
    BitRank() {}
    // 位向量长度
    void resize(const unsigned int num) {
        block.resize(((num + 1) >> 6) + 1, 0);
        count.resize(block.size(), 0);
    }
    // 设置i位bit
    void set(const unsigned int i, const unsigned long long val) {
        block[i >> 6] |= (val << (i & 63));
    }
    void build() {
        for (unsigned int i = 1; i < block.size(); i++) {
            count[i] = count[i - 1] + __builtin_popcountll(block[i - 1]);
        }
    }
    // [0, i) 1的个数
    unsigned int rank1(const unsigned int i) const {
        return count[i >> 6] + __builtin_popcountll(block[i >> 6] & ((1ULL << (i &
    63)) - 1ULL));
    }
    // [i, j) 1的个数
    unsigned int rank1(const unsigned int i, const unsigned int j) const {
        return rank1(j) - rank1(i);
    }
    // [0, i) 0的个数
    unsigned int rank0(const unsigned int i) const {
        return i - rank1(i);
    }
    // [i, j) 0的个数
    unsigned int rank0(const unsigned int i, const unsigned int j) const {
        return rank0(j) - rank0(i);
    }
};


class WaveletMatrix {
private:
    unsigned int height;
    std::vector<BitRank> B;
    std::vector<int> pos;
public:
    WaveletMatrix() {}
    WaveletMatrix(std::vector<int> vec) : WaveletMatrix(vec, *std::max_element(vec.
    begin(), vec.end()) + 1) {}
    // sigma: 字母表大小(字符串的话), 数字序列的话是数的种类
    WaveletMatrix(std::vector<int> vec, const unsigned int sigma) {
        height = (sigma == 1) ? 1 : (64 - __builtin_clzll(sigma - 1));
        B.resize(height), pos.resize(height);
        for (unsigned int i = 0; i < height; ++i) {
            B[i].resize(vec.size());
            for (unsigned int j = 0; j < vec.size(); ++j) {
                B[i].set(j, get(vec[j], height - i - 1));
            }
            B[i].build();
            auto it = stable_partition(vec.begin(), vec.end(), [&](int c) {
                return !get(c, height - i - 1);
            });
            pos[i] = it - vec.begin();
        }
    }

    int get(const int val, const int i) {
        return (val >> i) & 1;
    }

    // [l, r] 中val出现的频率
    int rank(const int l, const int r, const int val) {
        return rank(r, val) - rank(l - 1, val);
    }

    // [0, i] 中val出现的频率
    int rank(int i, int val) {
        ++i;
        int p = 0;
        for (unsigned int j = 0; j < height; ++j) {
            if (get(val, height - j - 1)) {
                p = pos[j] + B[j].rank1(p);
                i = pos[j] + B[j].rank1(i);
            } else {
                p = B[j].rank0(p);
                i = B[j].rank0(i);
            }
        }
      return i - p;
    }

    // [l, r] 中第k小, 只算非负数
    int kth(int l, int r, int k) {
        ++r;
        int res = 0;
        for (unsigned int i = 0; i < height; ++i) {
            const int j = B[i].rank0(l, r);
            if (j >= k) {
                l = B[i].rank0(l);
                r = B[i].rank0(r);
            } else {
                l = pos[i] + B[i].rank1(l);
                r = pos[i] + B[i].rank1(r);
                k -= j;
                res |= (1 << (height - i - 1));
            }
      }
      return res;
    }

    // [l,r] 在[a, b] 值域的数字个数 数组中只可有非负数
    int rangeFreq(const int l, const int r, const int a, const int b) {
        return rangeFreq(l, r + 1, a, b + 1, 0, 1 << height, 0);
    }
    int rangeFreq(const int i, const int j, const int a, const int b, const int l, const int r, const int x) {
        if (i == j || r <= a || b <= l) return 0;
        const int mid = (l + r) >> 1;
        if (a <= l && r <= b) {
            return j - i;
        } else {
            const int left = rangeFreq(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1);
            const int right = rangeFreq(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(j), a, b, mid, r, x + 1);
            return left + right;
        }
    }

    // [l,r] 在[a,b] 值域内存在的最小值是什么, 不存在返回-1, 只支持非负整数
    int rangeMin(int l, int r, int a, int b) {
        return rangeMin(l, r + 1, a, b + 1, 0, 1 << height, 0, 0);
    }
    int rangeMin(const int i, const int j, const int a, const int b, const int l, const int r, const int x, const int val) {
        if (i == j || r <= a || b <= l) return -1;
        if (r - l == 1) return val;
        const int mid = (l + r) >> 1;
        const int res = rangeMin(B[x].rank0(i), B[x].rank0(j), a, b, l, mid, x + 1, val);
        if (res < 0) {
            return rangeMin(pos[x] + B[x].rank1(i), pos[x] + B[x].rank1(j), a, b, mid, r, x + 1, val + (1 << (height - x - 1)));
        } else {
            return res;
        }
    }
};

void solve() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    WaveletMatrix tree(a);
    while(m--) {
        int l, r, k;
        cin >> l >> r >> k;
        cout << tree.kth(l, r, k) << '\n';
    }




}


int main() {
    std::ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

    int T = 1;
    //cin >> T;
    while (T--) solve();

    return 0;
}
```

1.8 笛卡尔树

```cpp
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
template<typename T>
struct Cartesian {
    Cartesian() = default;
    Cartesian(const std::vector<T> &v)
        : ls(v.size(), -1), rs(v.size(), -1) {
        std::stack<int> stk;
        for(int i = 1; i < v.size(); ++i) { // 维护下标 1~n
            while(!stk.empty() && v[i] < v[stk.top()]) {
                stk.pop();
            }
            if(stk.empty()) {
                ls[i] = root;
                root = i;
            } else {
                ls[i] = rs[stk.top()];
                rs[stk.top()] = i;
            }
            stk.push(i);
        }
    }
    int root = -1;
    std::vector<int> ls, rs;
};

void solve() {
    int n;
    cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];
    Cartesian tree(a);
    auto &ls = tree.ls, &rs = tree.rs;
    i64 ansl = 0, ansr = 0;
    for (int i = 1; i <= n; i++) {
        // cout << ls[i] << endl;
        ansl ^= 1ll * i * ((ls[i] == -1 ? 0 : ls[i]) + 1);
        ansr ^= 1ll * i * ((rs[i] == -1 ? 0 : rs[i]) + 1);
    }

    cout << ansl << ' ' << ansr;






}


int main() {
    std::ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

    int T = 1;
    //cin >> T;
    while (T--) solve();

    return 0;
}
```

1.9 线段树 (未修改) (SegmentTree+Info 初始赋值 + 单点修改 + 查找前驱后继)

```cpp
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

constexpr int inf = 1E9 + 1;
struct Info {
    int max = -inf;
    int min = inf;
};
Info operator+(const Info &a, const Info &b) {
    return { std::max(a.max, b.max), std::min(a.min, b.min) };
}
```

1.10 线段树（SegmentTree+Info 初始赋值 + 单点修改 + 查找前驱后继）

```cpp
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r + 1);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 1, n, l, r + 1, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r + 1, pred);
    }
};

constexpr int inf = 1E9 + 1;
struct Info {
    int max = -inf;
    int min = inf;
};
Info operator+(const Info &a, const Info &b) {
    return { std::max(a.max, b.max), std::min(a.min, b.min) };
}
```

2 图论

2.1 bellmanfloyd

```
#include <bits/stdc++.h>
using namespace std;
const int M = 1e4+1;
const int N = 550;
int n,m,k;
struct edge{
    int a,b,w;
}edges[M];
int dist[N];
int backup[N];
int bf(){

    memset(dist,0x3f,sizeof(dist));
    dist[1] = 0;
    for(int i = 1;i<=k;i++){
        memcpy(backup,dist,sizeof(dist));
        for(int j = 1;j<=m;j++){
            int a = edges[j].a;
            int b = edges[j].b;
            int w = edges[j].w;
            dist[b] = min(dist[b],backup[a]+w);
        }
    }
    if (dist[n] > 0x3f3f3f3f / 2) return -0x3f3f3f3f;
    else return dist[n];
}
int main()
{
    cin>>n>>m>>k;

    for(int i = 1;i<=m;i++){
        int a,b,w;
        cin>>a>>b>>w;
        edges[i] = {a,b,w};
    }

    int t = bf();

    if(t==-0x3f3f3f3f) cout<<"impossible"<<endl;
    else cout<<t<<endl;
    return 0;
}
```

2.2 dijk

```
#include <bits/stdc++.h>
using namespace std;
using pii = pair<int,int>;
const int N = 2e5+10;
bool st[N];
int main()
{
    int n,m;
    cin>>n>>m;
    int dist[N];
    memset(dist,0x3f,sizeof(dist));
    vector<vector<pii>> gra(n+1);
    while(m--){
        int a,b,c;
        cin>>a>>b>>c;
        gra[a].push_back({b,c});
    }
    priority_queue<pii,vector<pii>,greater<pii>> q;
    dist[1] = 0;
    q.push({0,1});
    while(!q.empty()){
        pii tnod = q.top();
        q.pop();
        int nod = tnod.second;
        int d = tnod.first;
        if(st[nod]) continue;
        st[nod] = true;
        for(auto [next_nod,vlen] : gra[nod]){
            if(dist[next_nod] > dist[nod]+vlen){
                dist[next_nod] = dist[nod]+vlen;
                q.push({dist[next_nod],next_nod});
            }
        }
    }
    if(dist[n]==0x3f3f3f3f) cout<<"-1"<<endl;
    else cout<<dist[n]<<endl;
    return 0;
}
```

2.3 dinic 求最大流

```
// 复杂度上界 O((n ^ 2) * m)
```

```cpp
// 求最小割容量即为最大流
// 最小割的最小化割边数量: 先跑一遍dinic, 把没有流满的边改为dinic, 流满的边改为
// 1, 重新跑一边dinic即为最小割边数量
// 没有最小割限制, 直接把所有边容量设为1, 跑一边dinic
template<class T>
struct MaxFlow {
    struct _Edge {
        int to;
        T cap;
        _Edge(int to, T cap) : to(to), cap(cap) {}
    };

    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g; //g存每个点连的边的e数组下标
    std::vector<int> cur, h; //h是点的深度

    MaxFlow() {}
    MaxFlow(int n) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        e.clear();
        g.assign(n, {});
        cur.resize(n);
        h.resize(n);
    }

    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {
                        return true;
                    }
                    que.push(v);
                }
            }
        }
        return false;
    }

    T dfs(int u, int t, T f) {
        if (u == t) {
            return f;
        }
        auto r = f;
        for (int &i = cur[u]; i < int(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            if (c > 0 && h[v] == h[u] + 1) {
                auto a = dfs(v, t, std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
        }
        return f - r;
    }
    void addEdge(int u, int v, T c) {
        g[u].push_back(e.size());
        e.emplace_back(v, c);
        g[v].push_back(e.size());
        e.emplace_back(u, 0);
    }
    T flow(int s, int t) { //求最大流
        T ans = 0;
        while (bfs(s, t)) {
            cur.assign(n, 0);
            ans += dfs(s, t, std::numeric_limits<T>::max());
        }
        return ans;
    }

    std::vector<bool> minCut() { //求一个最小割的划分, c[i] = 1在S集合, = 0在T集合
        std::vector<bool> c(n);
        for (int i = 0; i < n; i++) {
            c[i] = (h[i] != -1);
        }
        return c;
    }

    struct Edge {
        int from;
        int to;
        T cap;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

2.4 kruskal

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 1e5+5;
const int M = N*2;
int n,m;
int p[N];
struct edge{
    int a,b,w;
    bool operator < (const edge &W) const {
        return w<W.w;
    }
}edges[M];
int find(int x){
   if(p[x]!=x) p[x] = find(p[x]);
   return p[x];
}
int main()
{
    cin>>n>>m;
    for(int i = 1;i<=m;i++){
        int a,b,c;
        cin>>a>>b>>c;
        edges[i] = {a,b,c};
    }
    sort(edges+1,edges+1+m);
    int res = 0;
    int cnt = 0;
    for(int i = 1;i<=n;i++) p[i] = i;
    for(int i = 1;i<=m;i++){
        int a = edges[i].a;
        int b = edges[i].b;
        int c = edges[i].w;
        a = find(a);
        b = find(b);
        if(a==b){
            ;
        }else{
            p[a] = p[b];
            cnt++;
            res+=c;
        }
    }
    if(cnt!=n-1) cout<<"impossible";
    else cout<<res<<endl;
    return 0;
}
```

2.5 prim

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int N = 510;
const int M = 1e5+10;
int mp[N][N];
int dist[N];
bool st[N];
int n,m;

int prim(){
      memset(dist, 0x3f, sizeof dist);

    int res = 0;
    for (int i = 0; i < n; i ++ )
    {
        int t = -1;
        for (int j = 1; j <= n; j ++ )
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;

        if (i && dist[t] == 0x3f3f3f3f) return 0x3f3f3f3f;

        if (i) res += dist[t];
        st[t] = true;

        for (int j = 1; j <= n; j ++ ) dist[j] = min(dist[j], mp[t][j]);
    }

    return res;
}
int main()
{
    cin>>n>>m;
    memset(mp,0x3f,sizeof(mp));
    memset(dist,0x3f,sizeof(dist));
    for(int i = 1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        mp[u][v] = min(mp[u][v],w);
        mp[v][u] = min(mp[v][u],w);
    }
    int t = prim();
    if(t==0x3f3f3f3f) cout<<"impossible"<<endl;
    else cout<<t<<endl;
    return 0;
}
```

2.6 tarjan 割点

```cpp
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;


struct VBCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<vector<int>> ecc;
    vector<int> dfn, low;
    int cur, tot;
    VBCC(int n) {
        init(n);
    }

    void init(int n){
        this->n = n;
        adj.assign(n + 1, {});
        ecc.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.resize(n + 1);
        stk.clear();
        cur = tot = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    void dfs(int u, int p) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);
        int child = 0;
        for(auto v : adj[u]) {
            if(!dfn[v]) {
                child++;
                dfs(v, u);
                low[u] = min(low[u], low[v]);
                if(low[v] >= dfn[u]) {
                    ++tot;
                    while(1) {
                        int now = stk.back();
                        stk.pop_back();
                        ecc[tot].push_back(now);
                        if(now == v) break;
                    }
                    ecc[tot].push_back(u);
                }
            } else if(v != p) {
                low[u] = min(low[u], dfn[v]);
            }
        }

        if(p == -1 && child == 0) {
            ++tot;
            ecc[tot].push_back(u);
        }
    }

    vector<vector<int>> work() {
        for(int i = 1; i <= n; ++i) {
            if(!dfn[i]) {
                dfs(i, -1);
            }
        }
        return ecc;
    }


};



void solve(){
    int n, m;
    cin >> n >> m;
    VBCC g(n);
    for(int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }

    auto ecc = g.work();
    int tot = g.tot;

    cout << tot << '\n';

    for(int i = 1; i <= tot; i++) {
        cout << ecc[i].size() << ' ';
        for(auto x : ecc[i]) cout << x << ' ';
        cout << '\n';
    }







}


int main(){
    std::ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

    int T = 1;
    //cin >> T;
    while(T--) solve();

    return 0;
}
```

2.7 tarjan 求强连通分量

```cpp
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;
// 传n 使用下标 1~n
struct SCC {
    int n;
    vector<vector<int>> adj;
    vector<int> stk;
    vector<int> dfn, low, bel;
    // vector<vector<int>> ecc;
    int cur, tot;
    SCC(int n) {
        init(n);
    }

    void init(int n){
        this->n = n;
        adj.assign(n + 1, {});
        // ecc.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.resize(n + 1);
        bel.assign(n + 1, 0);
        stk.clear();
        cur = tot = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(v);
    }

    void dfs(int u) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);
        for(auto v : adj[u]) {
            if(!dfn[v]) {
                dfs(v);
                low[u] = min(low[u], low[v]);
            } else if(bel[v] == 0) {
                low[u] = min(low[u], low[v]);
            }
        }

        if(low[u] == dfn[u]) {
            ++tot;
            while(1) {
                int now = stk.back();
                // ecc[tot].push_back(now);
                bel[now] = tot;
                stk.pop_back();
                if(now == u) break;
            }
        }
    }

    vector<int> work() {
        for(int i = 1; i <= n; ++i) {
            if(!dfn[i]) {
                dfs(i);
            }
        }
        return bel;
    }


};



void solve(){

    int n, m;
    cin >> n >> m;

    vector<int> a(n + 1);
    vector<vector<int>> edge1(n + 1);
    for(int i = 1; i <= n; i++) cin >> a[i];
    SCC g(n);
    for(int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
        edge1[u].push_back(v);
    }

    auto bel = g.work();

    int tot = g.tot;
    vector<int> val(tot + 1), vis(tot + 1), ans(tot + 1);
    vector<vector<int>> edge2(tot + 1);

    for(int i = 1; i <= n; i++) {
        val[bel[i]] += a[i];
        for(auto j : edge1[i]) {
            if(bel[i] != bel[j]) edge2[bel[i]].push_back(bel[j]);
        }
    }
    int res = 0;
    auto dfs = [&] (auto self, int u, int p) -> void {
        if(vis[u]) return;
        vis[u] = p;
        for(auto v : edge2[u]) {
            if(vis[v] != u) {
                self(self, v, u);
                ans[u] = max(ans[u], ans[v]);
            }

        }
        ans[u] += val[u];
        res = max(res, ans[u]);
    };
    for(int i = 1; i <= tot; i++) {
        if(!vis[i]) {
            dfs(dfs, i, -1);
        }
    }
    cout << res << '\n';

}


int main(){
    std::ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

    int T = 1;
    //cin >> T;
    while(T--) solve();

    return 0;
}
```

2.8 tarjan 求边双连通分量

```cpp
#include<bits/stdc++.h>
using namespace std;
using i64 = long long;


struct EBCC {
    int n;
    vector<vector<int>> adj;
    vector<int> e;
    vector<int> stk;
    vector<int> dfn, low, bel;
    // vector<vector<int>> ecc;
    // vector<pair<int, int>> bridge; //割边
    int cur, tot;
    EBCC(int n) {
        init(n);
    }

    void init(int n){
        this->n = n;
        adj.assign(n + 1, {});
        // ecc.assign(n + 1, {});
        dfn.assign(n + 1, 0);
        low.resize(n + 1);
        bel.assign(n + 1, 0);
        stk.clear();
        e.clear();
        cur = tot = 0;
    }

    void addEdge(int u, int v) {
        adj[u].push_back(e.size());
        e.emplace_back(v);
        adj[v].push_back(e.size());
        e.emplace_back(u);
    }

    void dfs(int u, int laeid) {
        dfn[u] = low[u] = ++cur;
        stk.push_back(u);
        for(auto eid : adj[u]) {
            int v = e[eid];
            if(!dfn[v]) {
                dfs(v, eid);
                low[u] = min(low[u], low[v]);
            } else if(eid != (laeid ^ 1)) {
                low[u] = min(low[u], dfn[v]);
            }
        }

        if(low[u] == dfn[u]) {
            ++tot;
            // if(laeid != -1) bridge.push_back({u, e[laeid ^ 1]});
            while(1) {
                int now = stk.back();
                // ecc[tot].push_back(now);
                bel[now] = tot;
                stk.pop_back();
                if(now == u) break;
            }
        }
    }

    vector<int> work() {
        for(int i = 1; i <= n; ++i) {
            if(!dfn[i]) {
                dfs(i, -1);
            }
        }
        return bel; // bel : 每个点在哪个连通分量
    }


};
// 点下标为1~n是，传的参数n应该为n而不是n+1
// ecc可求出每个点在哪个连通分量 下标是 1~tot
// bel是每个点属于哪个连通分量 分量标号是1~tot
// tot是连通分量的个数
// bridge存的是每个割边的两个顶点



void solve(){
    int n, m;
    cin >> n >> m;
    EBCC g(n);

    for(int i = 1; i <= m; i++)  {
        int u, v;
        cin >> u >> v;
        g.addEdge(u, v);
    }

    auto bel = g.work();
    int tot = g.tot;
    cout << tot << '\n';

    vector<vector<int>> ebcc(tot + 1);

    for(int i = 1; i <= n; i++) {
        ebcc[bel[i]].push_back(i);
    }

    for(int i = 1; i <= tot; i++) {
        cout << ebcc[i].size() << ' ';
        for(auto it : ebcc[i]) cout << it << ' ';
            cout << '\n';
    }


```

```cpp




}


int main(){
    std::ios::sync_with_stdio(false), cin.tie(0), cout.tie(0);

    int T = 1;
    //cin >> T;
    while(T--) solve();

    return 0;
}
```

2.9 匈牙利算法

```cpp
int match[N]; //match是表示右边的点匹配左边的哪一个点
set<int> st; // 每次dfs后要清空
bool dfs(int x) {
    for(auto it : edge[x]) {
        if(st.count(it)) continue;
        st.insert(it);
        if(!match[it] || dfs(match[it])) {
            match[it] = x;
            return 1;
        }
    }


    return 0;
}
```

2.10 最大流最小费用

```cpp
template<class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;
        _Edge(int to_, T cap_, T cost_) : to(to_), cap(cap_), cost(cost_) {}
    };
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n, std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T, int>, std::vector<std::pair<T, int>>, std::
greater<std::pair<T, int>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d + h[u] - h[v] + cost) {
                    dis[v] = d + h[u] - h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
        return dis[t] != std::numeric_limits<T>::max();
    }
    MinCostFlow() {}
    MinCostFlow(int n_) {
        init(n_);
    }
    void init(int n_) {
        n = n_;
        e.clear();
        g.assign(n, {});
    }
    void addEdge(int u, int v, T cap, T cost) {
        g[u].push_back(e.size());
        e.emplace_back(v, cap, cost);
        g[v].push_back(e.size());
        e.emplace_back(u, 0, -cost);
    }
    std::pair<T, T> flow(int s, int t) { //返回first: 最大流量, second: 最小费用
        T flow = 0;
        T cost = 0;
        h.assign(n, 0);
        while (dijkstra(s, t)) {
            for (int i = 0; i < n; ++i) {
                h[i] += dis[i];
            }
            T aug = std::numeric_limits<int>::max();
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                aug = std::min(aug, e[pre[i]].cap);
            }
            for (int i = t; i != s; i = e[pre[i] ^ 1].to) {
                e[pre[i]].cap -= aug;
                e[pre[i] ^ 1].cap += aug;
            }
            flow += aug;
            cost += aug * h[t];
        }
        return std::make_pair(flow, cost);
    }
    struct Edge {
        int from;
        int to;
        T cap;
        T cost;
        T flow;
    };
    std::vector<Edge> edges() {
        std::vector<Edge> a;
        for (int i = 0; i < e.size(); i += 2) {
            Edge x;
            x.from = e[i + 1].to;
            x.to = e[i].to;
            x.cap = e[i].cap + e[i + 1].cap;
            x.cost = e[i].cost;
            x.flow = e[i + 1].cap;
            a.push_back(x);
        }
        return a;
    }
};
```

3 树上问题

3.1 LCA(倍增)

```cpp
vector<int> dep(tot + 1), vis1(tot + 1);
vector<array<int, 31>> anc(n + 1);

auto dfs1 = [&] (auto self, int u, int p) -> void{
    vis1[u] = 1;
    anc[u][0] = p;
    dep[u] = dep[p] + 1;


    for(int i = 1; i <= 30; i++) {
        anc[u][i] = anc[anc[u][i - 1]][i - 1];
    }

    for(auto v : edge[u]) {
        if(v == p) continue;
        self(self, v, u);
    }
};
//森林
for(int i = 1; i <= tot; i++) {
    if(!vis1[i]) dfs1(dfs1, i, 0);
}

auto lca = [&] (int u, int v) {
    int zu = u, zv = v;
    if(dep[zu] < dep[zv]) swap(zu, zv);

    int gap = dep[zu] - dep[zv];
    for(int i = 0; i <= 30; i++) {
        if((gap >> i) & 1) zu = anc[zu][i];
    }
    if(zu == zv) return zu;

    for(int i = 30; i >= 0; i--) {
        if(anc[zu][i] != anc[zv][i]) {
            zu = anc[zu][i];
            zv = anc[zv][i];
        }
    }
    return anc[zu][0];
};
```

3.2 LCA

```cpp
// 倍增求LCA
// 预处理O(nlogn) 单次查询O(logn)
void solve1(){
    int n, m, s;
    cin >> n >> m >> s;
    vector<vector<int>> edge(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        edge[u].push_back(v);
        edge[v].push_back(u);
    }

    vector<int> dep(n + 1);
    vector<vector<int>> anc(n + 1, vector<int>(31));
    auto dfs = [&] (auto self, int x, int f) -> void {
        dep[x] = dep[f] + 1;
        anc[x][0] = f;
        for(int i = 1; i < 31; i++) {
            anc[x][i] = anc[anc[x][i - 1]][i - 1];
        }

        for(auto y : edge[x]) {
            if(y == f) continue;
            self(self, y, x);
        }

    };
    dfs(dfs, s, 0);
    auto lca = [&] (int x, int y) -> int{
        if(dep[x] > dep[y]) swap(x, y);
        int gap = dep[y] - dep[x];
        for(int i = 0; i <= 30; i++) {
            if((gap >> i) & 1) y = anc[y][i];
        }
        if(x == y) return x;
        for(int i = 30; i >= 0; i--) {
            if(anc[x][i] != anc[y][i]) {
                x = anc[x][i];
                y = anc[y][i];
            }
        }
        return anc[x][0];

    };
    while(m--) {
        int x, y;
        cin >> x >> y;
        cout << lca(x, y) << '\n';
    }



}




//tarjan离线求LCA
//节点数n 查询数m O(n + m)
void solve2() {
    int n, m, s;
    cin >> n >> m >> s;
    vector<vector<int>> edge(n + 1);
    for(int i = 1; i < n; i++) {
        int u, v;
        cin >> u >> v;
        edge[u].push_back(v);
        edge[v].push_back(u);
    }
    vector<int> p(n + 1);
    for(int i = 1; i <= n; i++) p[i] = i;
    auto chazu = [&](auto self, int x) -> int {
        if(x != p[x]) p[x] = self(self, p[x]);
        return p[x];
    };
    vector<int> ans(m + 1), st(n + 1);
    vector<vector<info>> que(n + 1);
    for(int i = 1; i <= m; i++) {
        int a, b;
        cin >> a >> b;
        que[a].push_back({b, i});
        que[b].push_back({a, i});
    }

    auto dfs = [&] (auto self, int x, int f) -> void {
        st[x] = 1;

        for(auto y : edge[x]) {
            if(y == f) continue;
            self(self, y, x);
        }
        for(auto [y, idx] : que[x]) {
            if(st[y]) ans[idx] = chazu(chazu, y);
        }


        p[x] = f;


    };
    dfs(dfs, s, 0);

    for(int i = 1; i <= m; i++) cout << ans[i] << '\n';




}
```

3.3 重链剖分

```cpp
// 不能使用0下标
// 传n + 1 使用下标 1~n
// son如果没有重儿子则为0
struct HLD {
    HLD(const int &n) : n(n), edge(n) {
        fa = dep = son = sz = top = dfn = out = rnk = std::vector<int>(n);
    }

    void addEdge(const int &x, const int &y) {
        edge[x].push_back(y);
        edge[y].push_back(x);
    }

    void dfs1(int u, int p) {
        sz[u] = 1;
        fa[u] = p;
        dep[u] = dep[p] + 1;
        for(const auto &v : edge[u]) {
            if(v == fa[u]) continue;
            dfs1(v, u);
            sz[u] += sz[v];
            if(sz[son[u]] < sz[v]) {
                son[u] = v;
            }
        }
    }
    void dfs2(int u, int t, int &dfncnt) {
        dfncnt++;
        dfn[u] = dfncnt;
        rnk[dfncnt] = u;
        top[u] = t;
        if(son[u] == 0) return;
        dfs2(son[u], t, dfncnt);
        for(const auto &v : edge[u]) {
            if(v == fa[u] || v == son[u]) continue;
            dfs2(v, v, dfncnt);
        }
    }
    void work(int root = 1) {
        int dfncnt = 0;
        dfs1(root, 0);
        dfs2(root, root, dfncnt);
    }
    int lca(int u, int v) {
        while(top[u] != top[v]) {
            if(dep[top[u]] < dep[top[v]]) {
                std::swap(u, v);
            }
            u = fa[top[u]];
        }
        return (dep[u] < dep[v] ? u : v);
    }

    int dis(int x, int y) {
        return dep[x] + dep[y] - 2 * dep[lca(x, y)];
    }

    // int kth(int id, int k) {
    //  if(k > dep[id]) return 0;
    //  while(dep[id] - dep[top[id]] + 1 <= k) {
    //      k -= (dep[id] - dep[top[id]] + 1);
    //      id = fa[top[id]];
    //  }
    //  return rnk[dfn[id] - k];
    // }

    vector<vector<int>> edge;
    vector<int> fa, dep, son, sz, top, dfn, out, rnk;
    int n;
};
```

4 字符串

4.1 KMP

```cpp
int main()
{
    //P为模式串
    cin >> n >> p + 1 >> m >> s + 1;

    for (int i = 2, j = 0; i <= n; i ++ )
    {
        while (j && p[i] != p[j + 1]) j = ne[j];
        if (p[i] == p[j + 1]) j ++ ;
        ne[i] = j;
    }

    for (int i = 1, j = 0; i <= m; i ++ )
    {
        while (j && s[i] != p[j + 1]) j = ne[j];
        if (s[i] == p[j + 1]) j ++ ;
        if (j == n)
        {
            printf("%d ", i - n);
            j = ne[j];
        }
    }

    return 0;
}
```

4.2 hash string

```
#include <bits/stdc++.h>
#define uint uint64_t
#define int long long
using namespace std;

const uint mod = (1ull << 61) - 1;
mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<uint> dist(mod / 2, mod - 1);
const uint base = dist(rnd);
const int N = 2e5 + 5;
vector<uint> p(N);

uint add(uint a, uint b)
{
    a += b;
    if (a >= mod)
        a -= mod;
    return a;
}

uint mul(uint a, uint b)
{
    __uint128_t c = __uint128_t(a) * b;
    return add(c >> 61, c & mod);
}

uint query(const vector<uint> &h, int l, int r)
{
    return add(h[r], mod - mul(h[l - 1], p[r - l + 1]));
}

void init()
{
    p[0] = 1;
    for (int i = 1; i < N; i++)
        p[i] = mul(p[i - 1], base);
}

template<typename T>
vector<uint> build(vector<T> &s)
{
    vector<uint> hashed(s.size(), 0);
    for (int i = 1; i < s.size(); i++)
        hashed[i] = add(mul(hashed[i - 1], base), s[i]);
    return hashed;
}

void solve()
{

}

signed main()
{
    // ios::sync_with_stdio(false);
    // cout.tie(nullptr);
    // cin.tie(nullptr);
    int T = 1;
    // cin >> T;
    while (T--)
        solve();
    return 0;
}
```

4.3 trie

```

struct Trie {

    int tot;
    vector<vector<int>> nex;
    vector<int> cnt; // 以这个节点结尾的字符串的个数
    Trie() : nex(100001, vector<int>(26)), cnt(100001) {}
    Trie(int n, int m) : nex(n + 1, vector<int>(m)), cnt(n + 1) {}
    // n为树中最多会有多少个节点，m表示有多少种字符
    void insert(string s, int t = 1) {    // 插入字符串
        int p = 0;
        for (int i = 0; i < s.length(); i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) nex[p][c] = ++tot;    // 如果没有，就添加结点
            p = nex[p][c];
        }
        cnt[p] += t;
    }

    bool find(string s) {    // 查找字符串
        int p = 0;
        for (int i = 0; i < s.length(); i++) {
            int c = s[i] - 'a';
            if (!nex[p][c]) return 0;
            p = nex[p][c];
        }
        return cnt[p];
    }
};
```

5 数学

5.1 exgcd

```
ll exgcd(ll a, ll b, ll &x, ll &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll g = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return g;
}
```

5.2 millerRabin

```
//O(klog^3(n)), k = 7
#include <bits/stdc++.h>
using i64 = long long;

i64 qmi(i64 a, i64 b, i64 mod) {
    i64 res = 1;
    while(b) {
        if(b & 1) {
            res = (__int128)res * a % mod;
        }
        a = (__int128)a * a % mod;
        b >>= 1;
    }
    return res;
}

bool Minller(i64 n) {
    if(n <= 1 || n % 2 == 0) return (n == 2);
    i64 u = n - 1, k = 0;
    while(u % 2 == 0) u /= 2, ++k;
    static std::vector<i64> base = {2, 325, 9375, 28178, 450775, 9780504,
    1795265022};
    for(auto x : base) {
        i64 res = qmi(x, u, n);
        if(res == 0 || res == 1 || res == n - 1) continue;
        for(int i = 1; i <= k; ++i) {
            res = (__int128)res * res % n;
            if(res == n - 1) break;
            if(i == k) return false;
        }
    }
    return true;
}

void solve() {
    i64 x;
    std::cin >> x;
    std::cout << (Minller(x) ? "YES" : "NO") << '\n';
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

5.3 qmi

```
int qmi(int a, int b) {
    int res = 1;
    while(b) {
        if(b & 1) res = 1ll * res * a % mod;
        b >>= 1;
        a = 1ll * a * a % mod;

```

```cpp
    }
    return res;
}
```

5.4 欧拉筛 (和一个数质因数分解)

```cpp
struct LinearSieve
{
    int n;
    vector<int> minp;
    vector<int> primes;
    vector<int> phi;
    vector<int> mu;

    LinearSieve(int n = 2e6 + 5) : n(n), minp(n + 1), phi(n + 1), mu(n + 1)
    {
        for (int i = 2; i <= n; i++)
        {
            if (minp[i] == 0)
            {
                minp[i] = i;
                phi[i] = i - 1;
                mu[i] = -1;
                primes.push_back(i);
            }
            for (int p : primes)
            {
                if (i * p >= n || p > minp[i])
                    break;

                minp[i * p] = p;

                if (p == minp[i])
                {
                    phi[i * p] = p * phi[i];
                    mu[i * p] = 0;
                    break;
                }
                else
                {
                    phi[i * p] = (p - 1) * phi[i];
                    mu[i * p] = -mu[i];
                }
            }
        }

        phi[1] = 1;
        mu[1] = 1;
    }

    map<int, int> factorize(int x) {
        map<int, int> facts;

        if (x <= n)
        {
            while (x > 1)
            {
                facts[minp[x]]++;
                x /= minp[x];
            }

            return facts;
        }

        for (int p : primes)
        {
            if (p * p > x) break;

            if (x % p == 0)
            {
                int count = 0;
                while (x % p == 0)
                {
                    x /= p;
                    count++;
                }
                facts[p] = count;
            }
        }

        if (x > 1) facts[x] = 1;

        return facts;
    }

    bool is_prime(int x)
    {
        if (x < 2 || x > n) return false;
        return minp[x] == x;
    }
}
```

5.5 矩阵

```cpp
template<typename T>
struct Matrix {
    Matrix() : Matrix(0, 0) {};
    Matrix(int _n, int _m, T val = T{}) : n(_n), m(_m), mt(n * m, val){}
    Matrix(const std::initializer_list<std::initializer_list<T>> &v) : Matrix(v.size
     (), v.begin()->size()) {
        int i = 0;
        for(auto &row : v) {
            assert(row.size() == m);
            for(auto &x : row) {
                mt[i++] = x;
            }
        }
    }
    std::span<T> operator[](int idx) {
        return std::span<T>(mt.data() + idx * m, m);
    }
    std::span<const T> operator[](int idx) const {
        return std::span<const T>(mt.data() + idx * m, m);
    }
    friend Matrix<T> operator*(const Matrix<T>& A, const Matrix<T>& B) {
        assert(A.m == B.n);
        Matrix<T> MT(A.n, B.m);
        for(int i = 0; i < MT.n; ++i) {
            for(int j = 0; j < MT.m; ++j) {
                for(int k = 0; k < A.m; ++k) {
                    MT[i][j] = MT[i][j] + A[i][k] * B[k][j];
                }
            }
        }
        return MT;
    }
    friend Matrix<T> operator+(const Matrix<T>& A, const Matrix<T>& B) {
        assert(A.n == B.n && A.m == B.m);
        Matrix<T> MT(A.n, B.m);
        for(int i = 0; i < MT.n; ++i) {
            for(int j = 0; j < MT.m; ++j) {
                MT[i][j] = A[i][j] + B[i][j];
            }
        }
        return MT;
    }
    friend Matrix<T> operator-(const Matrix<T>& A, const Matrix<T>& B) {
        assert(A.n == B.n && A.m == B.m);
        Matrix<T> MT(A.n, B.m);
        for(int i = 0; i < MT.n; ++i) {
            for(int j = 0; j < MT.m; ++j) {
                MT[i][j] = A[i][j] - B[i][j];
            }
        }
        return MT;
    }
    static Matrix<T> eye(int n) {
        Matrix<T> MT(n, n);
        for(int i = 0; i < n; ++i) {
            MT[i][i] = 1;
        }
        return MT;
    }
    static Matrix<T> qpow(Matrix<T> a, i64 b) {
        Matrix<T> res(Matrix<T>::eye(a.n));
        while(b != 0) {
            if(b & 1) {
                res = res * a;
            }
            a = a * a;
            b >>= 1;
        }
        return res;
    }
    friend std::ostream& operator<<(std::ostream& os, const Matrix<T>& o) {
        for(int i = 0; i < o.n; ++i) {
            for(int j = 0; j < o.m; ++j) {
                os << o[i][j] << " \n"[j + 1 == o.m];
            }
        }
        return os;
    }
    int n, m;
    std::vector<T> mt;
};
```

5.6 线性筛

```cpp
vector<int> minp, primes;

void sieve(int n) {
    minp.assign(n + 1, 0);
    primes.clear();

    for (int i = 2; i <= n; i++) {
        if (minp[i] == 0) {
            minp[i] = i;
            primes.push_back(i);
        }

        for (auto p : primes) {
            if (i * p > n) {
                break;
            }
            minp[i * p] = p;
            if (p == minp[i]) {
                break;
            }
        }
    }
}

bool isprime(int n) {
    return minp[n] == n;
}
```

5.7 组合数

```cpp
vector<i64> fac, invfac;
int power(int a, int b) {
    int res = 1;
    while (b) {
        if (b % 2) res = 1LL * res * a % mod;
        a = 1LL * a * a % mod;
        b >>= 1;
    }
    return res;
}

void init(int n) {
    fac.resize(n + 1);
    invfac.resize(n + 1);

    fac[0] = 1;
    for (int i = 1; i <= n; i++) {
        fac[i] = 1LL * fac[i - 1] * i % mod;
    }
    invfac[n] = power(fac[n], mod - 2);  // 使用费马小定理求逆元
    for (int i = n - 1; i >= 0; i--) {
        invfac[i] = 1LL * invfac[i + 1] * (i + 1) % mod;
    }
}

// 计算组合数 C(n, m)
int binom(int n, int m) {
    if (n < m || m < 0) return 0;
    return 1LL * fac[n] * invfac[m] % mod * invfac[n - m] % mod;
}
int A(int n, int m) {
    return 1ll * fac[n] * invfac[n - m] % mod;
}
```

6 计算几何

6.1 凸包

```cpp
#include <bits/stdc++.h>
using i64 = long long;
constexpr long double EPS = 1E-10;
const long double PI = acos(-1.0);
using T = long double;
struct Point {
    T x = 0, y = 0;
    Point operator+(const Point &o) const {return {x + o.x, y + o.y};}
    Point operator-(const Point &o) const {return {x - o.x, y - o.y};}
    Point operator-() const {return {-x, -y};}
    Point operator*(T fac) const {return {x * fac, y * fac};}
    Point operator/(T fac) const {return {x / fac, y / fac};}
    bool operator<(const Point &o) const {
        return std::tie(x, y) < std::tie(o.x, o.y);
    }
    friend std::istream &operator>>(std::istream &is, Point &p) {
        return is >> p.x >> p.y;
    }
    friend std::ostream &operator<<(std::ostream &os, Point p) {
        return os << "(" << p.x << ", " << p.y << ")";
    }
};

struct Line {
    Point s, t;
    Line() = default;
    Line(Point _s, Point _t) : s(_s), t(_t) {}
};

int sgn(T a){  //判断正负
    if(fabs(a) < EPS) return 0;
    return a > 0 ? 1 : -1;
}

T dot(const Point &a, const Point &b) { // 计算点积
    return a.x * b.x + a.y * b.y;
}
T cross(const Point &a, const Point &b) { // 两向量叉积
    return a.x * b.y - a.y * b.x;
}
T cross(const Point &a, const Point &b, const Point &c) { // ab向量与ac向量的叉积
```

```cpp
    return cross(b - a, c - a);                        // c在直线ab左侧 > 0, 右侧 < 0, 共线 = 0
}
T len(const Point &a) { // 求模长
    return sqrtl(a.x * a.x + a.y * a.y);
}
T angle(const Point &a, const Point &b) { // 求夹角
    return acosl(dot(a, b) / len(a) / len(b)); //(弧度制) [0, π]
    return acosl(dot(a, b) / len(a) / len(b)) * (180 / PI); // 转化为角度制 [0, 180]
}
T dis2(const Point &a, const Point &b) { // 距离的平方
    return (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
}
T dis(const Point &a, const Point &b) { // 距离
    return sqrtl(dis2(a, b));
}
Point rotate(const Point &a, const Point &b, T theta) {  // 逆转角, ab向量逆时针旋转theta度, ac为转后向量,c为返回值
    return {
        (b.x - a.x) * cosl(theta) - (b.y - a.y) * sinl(theta) + a.x,
        (b.x - a.x) * sinl(theta) + (b.y - a.y) * cosl(theta) + a.y
    };
}

bool intersect(const Line &a, const Line &b) { //两线段相交返回1
    return cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t) <= 0
        && cross(b.s, b.t, a.s) * cross(a.s, b.t, a.t) <= 0;
}
bool intersectStrictly(const Line &a, const Line &b) { // 两线段严格相交
    return cross(a.s, a.t, b.s) * cross(a.s, a.t, b.t) < 0
        && cross(b.s, b.t, a.s) * cross(a.s, b.t, a.t) < 0;
}

Point getNode(const Line &a, const Line &b) { // 两直线交点
    Point u = a.t - a.s, v = b.t - b.s;
    T t = cross(a.s - b.s, v) / cross(v, u);
    return a.s + u * t;

};

std::vector<Point> andrew(std::vector<Point> &v) {
    int n = v.size();
    std::sort(v.begin(), v.end());
    std::vector<Point> stk;
    for(int i = 0; i < n; ++i) {
        while(stk.size() > 1 && cross(stk[stk.size() - 2], stk.back(), v[i]) <= 0) {
            stk.pop_back();
        }
        stk.push_back(v[i]);
    }
    int t = stk.size();
    for(int i = n - 2; i >= 0; --i) {
        while(stk.size() > t && cross(stk[stk.size() - 2], stk.back(), v[i]) <= 0) {
            stk.pop_back();
        }
        stk.push_back(v[i]);
    }
    stk.pop_back();
    return stk;
};

T diameter(const std::vector<Point> &v) {
    int n = v.size();
    T res = 0;
    for(int i = 0, j = 1; i < n; ++i) {
        while(sgn(cross(v[i], v[(i + 1) % n], v[j]) - cross(v[i], v[(i + 1) % n], v[(j + 1) % n])) <= 0) {
            j = (j + 1) % n;
        }
        res = std::max({res, dis(v[i], v[j]), dis(v[(i + 1) % n], v[j])});
    }
    return res;
}

T diameter2(const std::vector<Point> &v) {
    int n = v.size();
    T res = 0;
    for(int i = 0, j = 1; i < n; ++i) {
        while(sgn(cross(v[i], v[(i + 1) % n], v[j]) - cross(v[i], v[(i + 1) % n], v[(j + 1) % n])) <= 0) {
            j = (j + 1) % n;
        }
        res = std::max({res, dis2(v[i], v[j]), dis2(v[(i + 1) % n], v[j])});
    }
    return res;
}

T grith(const std::vector<Point> &convex) {
    long double ans = 0;
    for(int i = 0; i < convex.size(); ++i) {
        ans += dis(convex[i], convex[(i + 1) % convex.size()]);
    }
    return ans;
}

void solve() {
    int n, m;
    std::cin >> n;
    std::vector<Point> A(n);
    for(int i = 0; i < n; ++i) {
        std::cin >> A[i];
    }
    std::cin >> m;
    std::vector<Point> B(m);
    for(int i = 0; i < m; ++i) {
        std::cin >> B[i];
    }
    long double ans = grith(A) + 2.0L * sqrtl(diameter2(B)) * acosl(-1.0L);   //A周
     长 + 2 * B直径 * PI
    std::cout << std::fixed << std::setprecision(15) << ans << '\n';
}

int main(){
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int T = 1;
    std::cin >> T;
    while(T--) {
        solve();
    }
    return 0;
}
```

7 附录

7.1 VSCode 设置

VSCode 常用设置

- Auto Save (自动保存)
- Alt 键的快捷键
- Code Runner 运行快捷键
  - Run in Terminal
  - Run in Terminal
- 有时间可选: smooth

7.2 互质的规律

互质规律: 比较常见的定义 1. 较大数是质数, 两个数互质

2. 较小数是质数, 较大数不是它的倍数, 两个数互质

3. 1 与其他数互质

4. 2 与奇数互质

一些推论 1. 两个相邻的自然数一定互质

2. 两个相邻的奇数一定互质

3. n 与 2n + 1 或 2n - 1 一定互质

求差判断法如果两个数相差不大，可先求出它们的差，再看差与其中较小数是否互质。如果互质，则原来两个数一定是互质数。如：194 和 201，先求出它们的差，201 - 194 = 7，因 7 和 194 互质，则 194 和 201 是互质数。相反也成立, 对较大数也成立

求商判断法用大数除以小数，如果除得的余数与其中较小数互质，则原来两个数是互质数。如：317 和 52，317÷52 = 6......5，因余数 5 与 52 互质，则 317 和 52 是互质数。

7.3 常数表

n	$\log_{10} n$	$n!$	$C(n, n/2)$	LCM$(1...n)$	P_n
2	0.30102999	2	2	2	2
3	0.47712125	6	3	6	3
4	0.60205999	24	6	12	5
5	0.69897000	120	10	60	7
6	0.77815125	720	20	60	11
7	0.84509804	5040	35	420	15
8	0.90308998	40320	70	840	22
9	0.95424251	362880	126	2520	30
10	1.00000000	3628800	252	2520	42
11	1.04139269	39916800	462	27720	56
12	1.07918125	479001600	924	27720	77
15	1.17609126	1.31e12	6435	360360	176
20	1.30103000	2.43e18	184756	232792560	627
25	1.39794001	1.55e25	5200300	26771144400	1958
30	1.47712125	2.65e32	155117520	1.444e14	5604
P_n	37338_{40}	204226_{50}	966467_{60}	190569292_{100}	$1e9_{114}$

$\max \omega(n)$：小于等于 n 中的数最大质因数个数

$\max d(n)$：小于等于 n 中的数最大因数个数

$\pi(n)$：小于等于 n 中的数最大互质数个数

$n \le$	10	100	1e3	1e4	1e5	1e6
$\max \omega(n)$	2	3	4	5	6	7
$\max d(n)$	4	12	32	64	128	240
$\pi(n)$	4	25	168	1229	9592	78498
$n \le$	1e7	1e8	1e9	1e10	1e11	1e12
$\max \omega(n)$	8	8	9	10	10	11
$\max d(n)$	448	768	1344	2304	4032	6720
$\pi(n)$	664579	5761455	5.08e7	4.55e8	4.12e9	3.7e10
$n \le$	1e13	1e14	1e15	1e16	1e17	1e18
$\max \omega(n)$	12	12	13	13	14	15
$\max d(n)$	10752	17280	26880	41472	64512	103680
$\pi(n)$	Prime number theorem: $\pi(x) \sim \frac{x}{\log(x)}$					

7.4 常见错因

爆数据 (爆 int, 爆 longlong)
取 mod 没有取干净或者取 mod 时超范围
想不出题事算一下各种数据范围

7.5 斐波那契数列

1. $\sum_{i=1}^{n} F_i \ = \ F_{n+2} - 1.$
2. $\sum_{i=1}^{n} F_{2i-1} \ = \ F_{2n}.$
3. $\sum_{i=1}^{n} F_{2i} \ = \ F_{2n+1} - 1.$
4. $\sum_{i=1}^{n} F_i^2 \ = \ F_n F_{n+1}.$
5. $F_{n+m} \ = \ F_{n+1}F_m \ + \ F_n F_{m-1}.$
6. $F_{n-1}F_{n+1} - F_n^2 \ = \ (-1)^n.$ (Cassini 恒等式)
7. $F_{2n-1} \ = \ F_n^2 + F_{n-1}^2.$
8. $F_n \ = \ \dfrac{F_{n+2} + F_{n-2}}{3}.$
9. $F_{2n} \ = \ F_n\big(F_{n+1} + F_{n-1}\big).$
10. 对任意 $k \in \mathbb{N}$，有 $F_n \mid F_{nk}$.
11. 若 $F_a \mid F_b$，则 $a \mid b$.
12. $\gcd(F_n, F_m) \ = \ F_{\gcd(n,m)}.$
13. $F_n = \dfrac{1}{\sqrt{5}}\left[\left(\dfrac{1+\sqrt{5}}{2}\right)^n - \left(\dfrac{1-\sqrt{5}}{2}\right)^n\right].$

7.6 算法

出现的错误检查

1. 超出范围 (int, i64)
2. 取模出现错误
3. 图可能有重边或者自环

杂项 (通用)

想不到的题考虑:

1. 二分, 二分答案, 三分
2. dp
3. 离线
4. 倒推
5. 倍增
6. 建图

分块

一个二进制状态, 枚举他的所有子集状态 `for(j = status; j > 0; j = (j - 1) & status);`

中位数:

- 变成二分变成 +1 -1 判断
- 若动态维护, 每次加入一个数然后求中位数, 用对顶堆
- 如果还有删除一个数, 那么就用对顶 multiset

DP

- 背包
- 区间 dp
- DAG 图上的 dp
- 树形 dp
  - 直接 dfs
  - DFN 序:
    - ∗ 能够知道这个节点的子树大小
    - ∗ 某节点在不在此子树
    - ∗ 适用于对于一个节点向一个还没有合并过的节点合并复杂度较低的问题

  - 换根 dp
  - 基环树
- 状压 dp
- 数位 dp

搜索

- dfs, bfs
- bfs 不适用于有权图 01bfs 可适用于权值只有 0 和 1 的图
- 双向广搜: 两边各搜一半再合并

图论

- **拓扑排序:**
  - 能够处理环, 算出环的大小, 链的大小等
- **tarjan 缩点 (连通性相关)**
  - 边双联通分量: 等价于: 该子图中任意两点之间至少存在两条边互不相交的路径
  - 点双联通分量: 等价于: 任意两点之间至少存在两条内部点互不相交的路径。

树

- **倍增**
  - lca
- **树的重心**
  - 定义:
    1. 以某个节点为根时，最大子树的节点数最少，那么这个节点是重心
    2. 以某个节点为根时，每颗子树的节点数不超过总节点数的一半，那么这个节点是重心
    3. 以某个节点为根时，所有节点都走向该节点的总边数最少，那么这个节点是重心
  - 性质:
    1. 一棵树最多有两个重心，如果有两个重心，那么两个重心一定相邻
    2. 如果树上增加或者删除一个叶节点，转移后的重心最多移动一条边
    3. 如果把两棵树连起来，那么新树的重心一定在原来两棵树重心的路径上
    4. 树上的边权如果都为正数，不管边权怎么分布，所有节点都走向重心的总距离和最小
- **树的直径**
  - 求法:
    1. 两次 dfs 找两个距离最远的点不适用于有负边的树
    2. 树形 dp 对于每个点找子树中的最长的两条链适用于所有树
  - 性质:
    如果树上的边权都为正，则有如下直径相关的结论：
    1. 如果有多条直径，那么这些直径一定拥有共同的中间部分，可能是一个公共点或一段公共路径
    2. 树上任意一点，相隔最远的点的集合，直径的两端点至少有一个在其中
- 树上差分
  - 点差分
  - 边差分
- 换根 dp
- 重链剖分
- 树上启发式合并
  - 适用于对多个子树统计答案
  - 树上启发式合并的特征：
    1. 没有修改操作
    2. 可以通过遍历子树，建立信息统计，得到所有查询的答案

数学

- 数论分块
- 互质的情况
- 素数密度
- **质数判断:**
  1. 一个较小质数判断, 试除法
  2. 一个较大质数判断, miller rabin
  3. 一个范围内质数 (较多质数) 判断欧拉筛
- **质因数分解:**
  1. 数量少用试除法 $O(\sqrt{n})$
  2. 数量多欧拉筛除以 minp

字符串

- kmp
- 字符串哈希

- trie

数据结构

- 链表, 栈, 队列
- **单调栈:** 能够知道每个数前面距离最近的比它大或者小的数
- **单调队列:** 能够知道每个长度为 k 的子区间的最值
- **堆**
  - 对顶堆: 能够方便的动态维护集合中第 k 大的元素
- **并查集:**
  - 扩展域/种类并查集: 能够维护满足 1. 朋友的朋友是朋友 2. 敌人的敌人是朋友
  - 带权并查集: 维护到根的距离并且取模能够达到类似扩展域并查集的效果
- **线段树:**
  - 区间加减
  - 区间修改 Tag 设一个 ip 变量代表是否修改
  - 势能分析直接暴力修改到叶子
  - 区间合并
  - 扫描线的修改, 区间懒标记是否被选取, 因为删除的时候只会删除已经添加过的区间
  - 动态开点
- **树状数组**
  - 能够维护可差分信息
  - 能够更快的边维护边查单个点的前缀
  - 能够动态地知道每个数前面有多少个小于它的数
  - kth 知道第 k 大的数是多少
  - 树上二分
- **波纹疾走树**
  - 查询区间第 k 小的数字
  - 区间内一个数字/一段数字的频率

7.7 组合数学公式

性质 1:

$$\mathrm{C}_{\mathrm{n}}^{\mathrm{m}} = \mathrm{C}_{\mathrm{n}}^{\mathrm{n-m}}$$

性质 2:

$$\mathrm{C}_{\mathrm{n+m+1}}^{\mathrm{m}} = \sum_{i=0}^{\mathrm{m}} \mathrm{C}_{\mathrm{n}+i}^{i}$$

性质 3:

$$\mathrm{C}_{\mathrm{n}}^{\mathrm{m}} \cdot \mathrm{C}_{\mathrm{m}}^{\mathrm{r}} = \mathrm{C}_{\mathrm{n}}^{\mathrm{r}} \cdot \mathrm{C}_{\mathrm{n-r}}^{\mathrm{m-r}}$$

性质 4 (二项式定理):

$$\sum_{i=0}^{\mathrm{n}} \left(\mathrm{C}_{\mathrm{n}}^{i} \cdot x^{i}\right) = (1+x)^{\mathrm{n}}$$

$$\sum_{i=0}^{\mathrm{n}} \mathrm{C}_{\mathrm{n}}^{i} = 2^{\mathrm{n}}$$

性质 5:

$$\sum_{i=0}^{\mathrm{n}} \left((-1)^{i} \cdot \mathrm{C}_{\mathrm{n}}^{i}\right) = 0$$

性质 6:

$$\mathrm{C}_{\mathrm{n}}^{0} + \mathrm{C}_{\mathrm{n}}^{2} + \cdots = \mathrm{C}_{\mathrm{n}}^{1} + \mathrm{C}_{\mathrm{n}}^{3} + \cdots = 2^{\mathrm{n}-1}$$

性质 7:

$$\mathrm{C}_{\mathrm{n+m}}^{\mathrm{r}} = \sum_{i=0}^{\min(\mathrm{n},\mathrm{m},\mathrm{r})} \left(\mathrm{C}_{\mathrm{n}}^{i} \cdot \mathrm{C}_{\mathrm{m}}^{\mathrm{r}-i}\right)$$

$$\mathrm{C}_{\mathrm{n+m}}^{\mathrm{n}} = \mathrm{C}_{\mathrm{n+m}}^{\mathrm{m}} = \sum_{i=0}^{\min(\mathrm{n},\mathrm{m})} \left(\mathrm{C}_{\mathrm{n}}^{i} \cdot \mathrm{C}_{\mathrm{m}}^{i}\right), \quad (\mathrm{r} = \mathrm{n} \mid \mathrm{r} = \mathrm{m})$$

性质 8:

$$\mathrm{m} \cdot \mathrm{C}_{\mathrm{n}}^{\mathrm{m}} = \mathrm{n} \cdot \mathrm{C}_{\mathrm{n-1}}^{\mathrm{m-1}}$$

性质 9:

$$\sum_{i=0}^{\mathrm{n}} \left(\mathrm{C}_{\mathrm{n}}^{i} \cdot i^{2}\right) = \mathrm{n}(\mathrm{n}+1) \cdot 2^{\mathrm{n}-2}$$

性质 10:

$$\sum_{i=0}^{\mathrm{n}} \left(\mathrm{C}_{\mathrm{n}}^{i}\right)^{2} = \mathrm{C}_{2\mathrm{n}}^{\mathrm{n}}$$

7.8 编译参数

-D_GLIBCXX_DEBUG : STL debugmode
-fsanitize=address : 内存错误检查
-fsanitize=undefined :UB 检查

7.9 运行脚本

Linux 运行脚本

```
#!/bin/bash
g++ -std=c++20 -O2 -Wall "$1.cpp" -o "$1" -D_GLIBCXX_DEBUG
./"$1" < in.txt > out.txt
cat out.txt
```

7.10 随机素数

979345007 986854057502126921

935359631 949054338673679153

931936021 989518940305146613

984974633 972090414870546877

984858209 956380060632801307